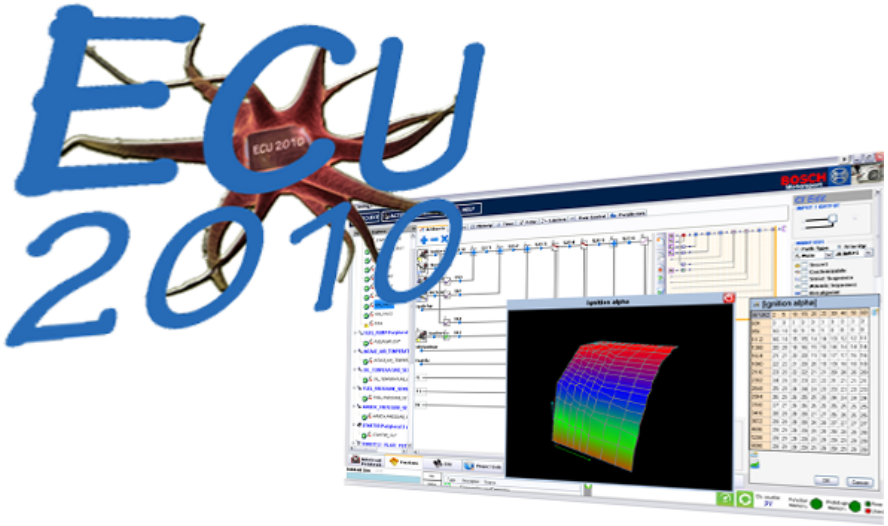
Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática,
2010

Nelson Carvalho Bernardino

Sistema de Desenvolvimento e Controlo para ECUs de Desporto Automóvel

Integrated Development and Management System for Motorsport ECUs



Universidade de Aveiro
Departamento de Electrónica, Telecomunicações e Informática,
2010

Nelson Carvalho Bernardino

Sistema de Desenvolvimento e Controlo para ECUs de Desporto Automóvel

Integrated Development and Management System for Motorsport ECUs

Dissertação apresentada à Universidade de Aveiro, para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia de Electrónica e Telecomunicações, realizada sob a orientação científica de Prof. Doutor Manuel Bernardo Salvador Cunha, Professor Auxiliar do Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro e de Prof. Doutor José Luis Costa Pinto de Azevedo, Professor Auxiliar no Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro.

o júri

Presidente	**Prof. Doutor Tomás Oliveira e Silva** Professor Associado da Universidade de Aveiro
Arguente Principal	**Prof. Doutor Sérgio Adriano Fernandes Lopes** Professor Auxiliar no Departamento de Electrónica Industrial da Esc. de Engenharia da Universidade do Minho
Orientador	**Prof. Doutor Manuel Bernardo Salvador Cunha** Professor Auxiliar no Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro
Co-Orientador	**Prof. Doutor José Luis Costa Pinto Azevedo** Professor Auxiliar no Departamento de Electrónica, Telecomunicações e Informática da Universidade de Aveiro

acknowledgements / agradecimentos

Agradeço à minha família, os meus pais Júlio e Fernanda e a minha irmã Simone, que me apoiaram a prosseguir os meus estudos e sempre me ajudaram incondicionalmente. À Carla, pelo constante apoio, preocupação e por me incentivar mesmo quando a vontade esmorecia. Aos meus colegas e amigos neste projecto, Filipe Teixeira, Rui Gomes e Paulo Martins, pelo excelente ambiente de trabalho, amizade e ajuda prestada. Agradeço ao Professor Doutor Bernardo Cunha pela coordenação e ideias sempre bem fundamentadas, pela perspectiva abrangente, inovadora e cabeça fria para encontrar sempre soluções. Ao nosso mentor, Pedro Kulzer, por ter acreditado em mim para fazer parte da equipa que escolheu para tornar realidade o seu projecto.

Keywords Graphic Programming, Motorsports, Live-debugg

Abstract The Motorsport Industry leads the path to the technological evolution of our day to day vehicles.
The development process requires the use of top-notch software and hardware.
This thesis presents the software component of a revolutionary approach to create an development and management environment that tackles all the current shortcomings of the industry, providing flexible and scalable tools and components.
The software created has the capability to: visually create, edit and interact with functions that in turn will be deployed to the hardware created in the one of sibling projects; control the hardware (directly and/or remotely) in a transparent and seamless manner; programming without compiler or linkers (the functions are directly interpreted by the system); debugging and changing the functions live, without the need to stop the hardware; store and analyse data logs; interact with intelligent peripherals, adding, removing, creating functions, reading and writing it's variables.

Palavras-Chave Programação Gráfica, Desporto Automóvel, Correcção de erros em tempo-real

Resumo A Indústria do Desporto Automóvel lidera a evolução tecnológica dos veículos do dia-a-dia.
O processo de desenvolvimento requer o uso de software e hardware de topo.
Esta dissertação apresenta a componente de software numa abordagem revolucionária para criar um ambiente de desenvolvimento e gestão que visa suprimir muitas das falhas e limitações presentes nesta indústria, fornecendo ferramentas e componentes flexíveis e escaláveis.
O software criado tem a capacidade de: duma forma visual, criar, editar e interagir com funções que, por sua vez, serão descarregadas para o hardware criado num dos projectos realizados em paralelo com este; controlar o hardware (directamente ou de forma remota) de forma transparente e fluída; programaço sem compiladores ou linkers (as funções são directamente interpretadas pelo sistema); correcção de erros e modificação de funções em tempo-real, sem ser necessário parar o hardware; armazenar e analisar dados; interagir com periféricos inteligentes, adicionando, removendo, criando funções, lendo e escrevendo as sua variáveis.

Contents

List of Figures

Chapter 1

Introduction

This work is part of a larger project code named "ECU2010 - Revolutionary Motorsports" and focuses in the software layer in a new approach to Automobile Electronic Control Units development and management, especially for the motorsport industry.

1.1 Motivation

The need for an evolution was felt by Engineer Pedro Kulzer, while he was working at Bosch Motorsport in Stuttgart, Germany. He made notes of all the problems and situations he wanted to avoid or improve, in is own experience and related to some feedback received by other motorsport coworkers and clients. The solution that he concocted involves a new hardware and software architecture so that all the shortcomings that have been identified in the present development environment in the motorsport industry can be overcome. The hardware must be scalable and flexible and the software must aggregate all the needed functionalities in one seamless package.

1.2 Objectives

It is the objective of this project to create a single software that, in symbiosis with the Cellular ECUs and Intelligent Peripherals developed in parallel, can address all the issues identified in the present motorsport tools while providing a seamless development environment.

This work should not be taken separately from the creations of the other elements in this project. It's the solution as a whole that makes sense and results in an interesting approach to the motorsport ECU devolopment and management.

The software shall:

- Work in the Microsoft Windows operative system;
- Aggregate all functionalities;
- Consist of a single executable file;
- Enable programming without compiler or linker;
- The resulting programming language should be directly interpreted by the system;
- Permit live codding and debugging (no need to stop the system to make changes);
- Perform management tasks for the complete system;
- Easy to understand and start working with;
- Present all possible actions without the need of memorising complicated menu structures;
- Have rollback possibilities;
- Save the complete ECU project in a single XML (eXtensible Markup Language) [10] [11] file;
- Portable to mobile devices (PDAs).

1.3 Thesis Structure

This thesis is structured as follows:

In chapter 2, is presented an overview of the state of the art of the software used in the motorsport industry.

In chapter 3 the possible solutions are presented and the selected one is explained.

Chapter 4 is where the actual work is described.

In chapter 5 exposes the results obtained at the end of the project.

Finally, chapter 6 discusses the results achieved and presents the conclusion and suggestions for future work.

Chapter 2

State Of The Art

In this area (Motorsport), there is no global solution for developing and managing all the project. For each function, different software is used and the user has to be the connection mechanism so that all the information obtained in one of the steps can be used in the subsequent actions performed in other software tools.

2.1 Matlab Simulink

For graphical and data-flow design, the Matlab Simulink by Mathworks [8] is one of the major players in the market.

The Matlab Simulink is a development environment used in model-based design and simulation for dynamic and embedded systems. This software permits to graphically, and using a block of libraries that can be customizable, design, test, implement and simulate various types of systems such as: signal, video and image processing, controls and communications.

Matlab Simulink is a commercial software and wasn't developed with motorsports in mind, however, it's ability to integrate add-on products extend Simulink software to multiple modeling domains, as well as provide tools for design, implementation, verification and validation tasks.

Simulink is integrated with MATLAB, providing immediate access to an extensive range of tools that let you develop algorithms, analyse and visualize simulations, create batch processing scripts, customize the modelling environment, and define signal, parameter, and test data.

Key Features:

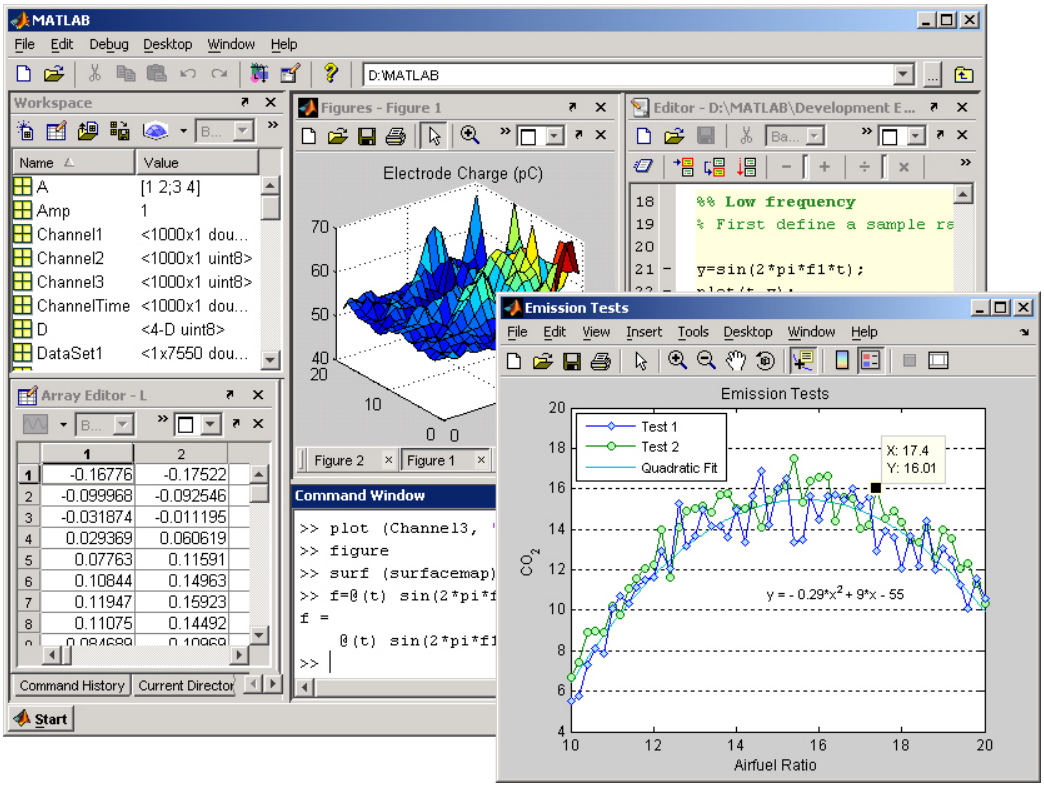


Figure 2.1: Matlab work environment.

- Interactive graphical editor for creating and managing block diagrams
- Extensive and expandable libraries of predefined blocks
- Ability to manage complex designs by segmenting models into hierarchies of design components
- Model Explorer to navigate, create, configure, and search all signals, parameters, properties, and generated code associated with your model
- Application programming interfaces (APIs) that let you connect with other simulation programs and incorporate hand-written code
- Embedded MATLAB Function blocks for bringing MATLAB algorithms into Simulink and embedded system implementations
- Simulation modes (Normal, Accelerator, and Rapid Accelerator) for running simulations interpretively or at compiled C-code speeds using fixed- or variable-step solvers
- Graphical debugger and profiler to examine simulation results and then diagnose performance and unexpected behaviour in your design
- Access to MATLAB for analysing and visualizing results, customizing the modelling environment, and defining signal, parameter, and test data
- Model analysis and diagnostics tools to ensure model consistency and identify modelling errors

2.2 Labview

Developed by National Instruments [7], Labview is a tool with focus on data-acquisition and instruments and equipment control.

Labview presents a visual programming language designated "G", auto-code generation for FPGAs and micro-controllers, graphical comparison tool.

Like Simulink, it's for general purpose and not designed specifically to motorsports.

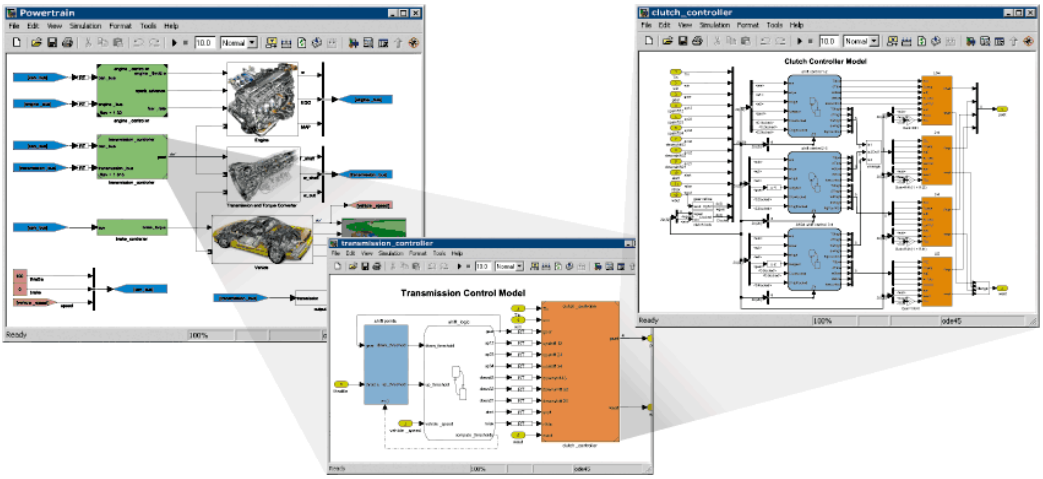


Figure 2.2: Simulink.

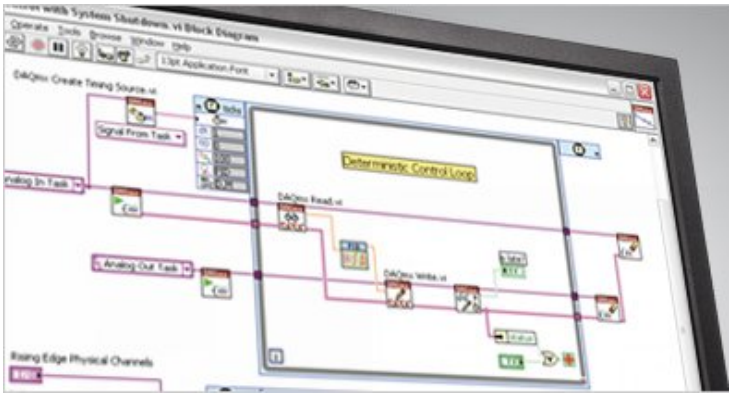


Figure 2.3: Labview.

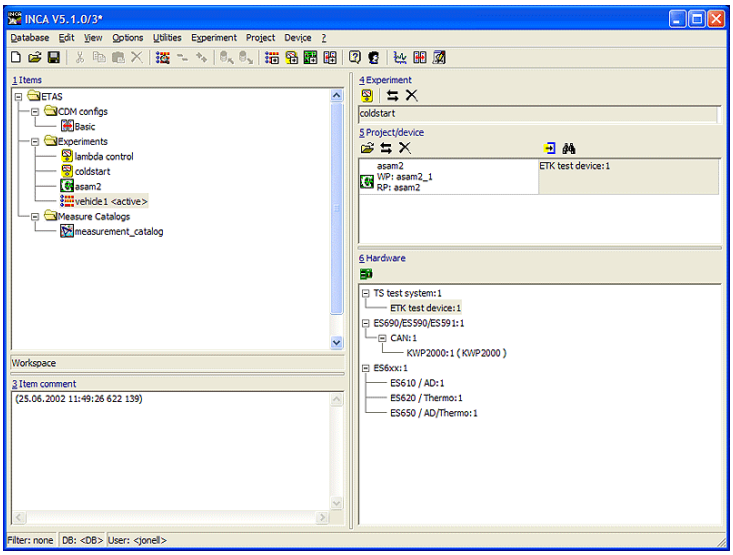


Figure 2.4: INCA.

2.3 INCA

The ETAS Group provides tools and tool solutions for the development and service of automotive ECUs. With more than 700 employees, the ETAS Group maintains offices in 11 locations around the globe. The base product INCA comprises the system core with its measurement and calibration functionality and tools for managing the configurations of ECU projects and calibration parameters, for analyzing measured data, and for reprogramming ECU software into flash memory.

INCA is able to:

- Monitor and record measurement signals of ECUs
- Monitor, change and record calibration parameters of ECUs Monitor and record analog signals from the vehicle environment
- Monitor and record digital signals from vehicle bus systems
- Send messages to the CAN bus in order to acquire analog measurement signals from the vehicle.

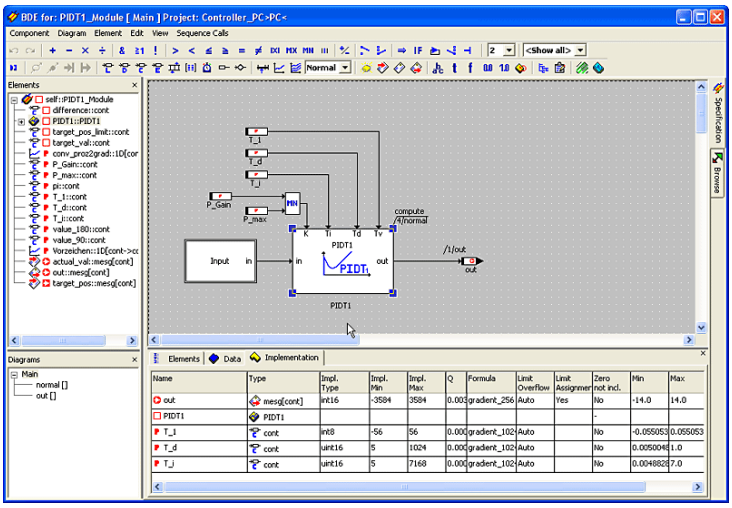


Figure 2.5: ASCET Workspace.

2.4 ASCET

Model components can be specified on the physical level with block diagrams, state machines, conditional tables, Boolean tables or textually, using ESDL or C language. The underlying object-based model architecture allows to flexibly combine model components specified with these different notations, and to build up models hierarchically. Once defined, model components can be used in different projects [2] [3].

To facilitate the cooperation between different engineering disciplines or vehicle manufacturers and suppliers, several import and export functions for model components on all hierarchical levels are provided.

ASCET-MD provides the functionality required to support the engineering tasks in a modular, thorough and efficient manner. Several features, designed to visualize the interdependencies in a model, assist users in keeping focus on their main task: the specification and design of control and diagnostic algorithms.

These features are:

- Block diagrams
- State machines

- Operating system specification
- Document generator
- Experiment environment
- Integrated code generator for floating-point and fixed-point arithmetic

Chapter 3

Possible Solutions Analysis

3.1 Programming Language

3.1.1 Borland Delphi

As a recommendation from the project's Mentor, Pedro Kulzer, one of the first options approached was to develop in Borland Delphi.
Borland Delphi is an integrated development environment, compiler and programming language used to create applications for Microsoft Windows and the .NET framework[5]. It was also possible to develop for Linux and Mac OS using an additional IDE - Kylix.
The reasons for the initial preference were:

- Quick compiling;
- Reliable tools;
- .NET capabilities.

At the time Delphi was property of Borland, however, today it's owned by Embarcadero Technologies [4].

3.1.2 Microsoft .NET Framework

.NET Framework Conceptual Overview

The .NET Framework is a Microsoft Windows component that supports building and running applications and XML Web services. The .NET Frame-

work was created to[9]:

- Achieve a consistent object-oriented programming environment either if the object code is stored and executed locally, executed locally but Internet-distributed or executed remotely;
- Present a code-execution environment that minimizes software deployment and versioning conflicts.
- Provide a code-execution environment that promotes safe execution of code, including code created by an unknown or semi-trusted third party.
- To provide a code-execution environment that eliminates the performance problems of scripted or interpreted environments.
- To make the developer experience consistent across widely varying types of applications, such as Windows-based applications and Web-based applications.
- To build all communication on industry standards to ensure that code based on the .NET Framework can integrate with other code.

The two main components of the .NET Framework are:

- The common language runtime (CLR) is the basis of the .NET Framework. The runtime can be viewed as an agent that manages code at execution time, providing core services such as memory management, thread management, and remoting, while also enforcing strict type safety and other forms of code accuracy that promote security and robustness. In fact, the concept of code management is a fundamental principle of the runtime. Code that targets the runtime is known as managed code, while code that does not target the runtime is known as unmanaged code.

- The .NET Framework class library, is an object-oriented collection of reusable types that you can use to develop applications ranging from traditional command-line or graphical user interface (GUI) applications to applications based on ASP.NET, such as Web Forms and XML Web services.

The .NET Framework can be hosted by unmanaged components that load the common language runtime into their processes and initiate the execution of managed code, thereby creating a software environment that can exploit both managed and unmanaged features. The .NET Framework not only provides several runtime hosts, but also supports the development of third-party runtime hosts.

For example, ASP.NET hosts the runtime to provide a scalable, server-side environment for managed code. ASP.NET works directly with the runtime to enable ASP.NET applications and XML Web services, both of which are discussed later in this topic.

The Figure 3.1 shows the relationship of the common language runtime and the class library to the applications and to the overall system. The illustration also shows how managed code operates within a larger architecture.

Common Language Runtime

The common language runtime manages memory, thread execution, code execution, code safety verification, compilation, and other system services. These features are intrinsic to the managed code that runs on the common language runtime.

With regards to security, managed components are awarded varying degrees of trust, depending on a number of factors that include their origin (such as the Internet, enterprise network, or local computer). This means that a managed component might or might not be able to perform file-access operations, registry-access operations, or other sensitive functions, even if it is being used in the same active application.

The runtime enforces code access security. For example, users can trust that an executable embedded in a Web page can play an animation on screen or sing a song, but cannot access their personal data, file system, or network. The security features of the runtime thus enable legitimate Internet-deployed software to be exceptionally feature rich.

The runtime also enforces code robustness by implementing a strict type-and-code-verification infrastructure called the common type system (CTS).

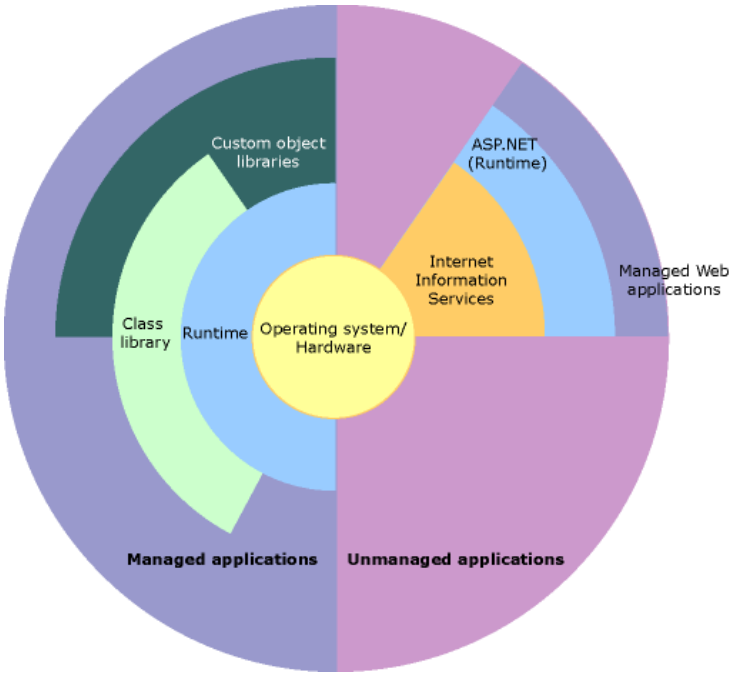


Figure 3.1: Common Language Runtime.

The CTS ensures that all managed code is self-describing. The various Microsoft and third-party language compilers generate managed code that conforms to the CTS. This means that managed code can consume other managed types and instances, while strictly enforcing type fidelity and type safety.

In addition, the managed environment of the runtime eliminates many common software issues. For example, the runtime automatically handles object layout and manages references to objects, releasing them when they are no longer being used. This automatic memory management resolves the two most common application errors, memory leaks and invalid memory references.

The runtime also accelerates developer productivity. For example, programmers can write applications in their development language of choice, yet take full advantage of the runtime, the class library, and components written in other languages by other developers. Any compiler vendor who chooses to target the runtime can do so. Language compilers that target

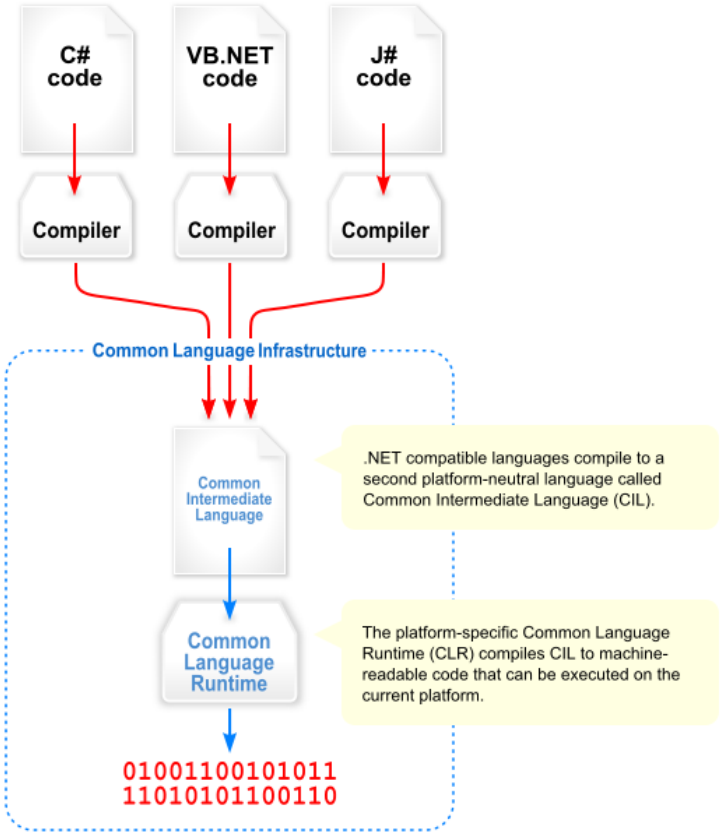


Figure 3.2: Overview of the Common Language Infrastructure.

the .NET Framework make the features of the .NET Framework available to existing code written in that language, greatly easing the migration process for existing applications.

The Runtime is designed to also supports software of today and yesterday. Interoperability between managed and unmanaged code enables developers to continue to use necessary COM components and DLLs.

The runtime is designed to enhance performance. Although the common language runtime provides many standard runtime services, managed code is never interpreted. A feature called just-in-time (JIT) compiling enables all managed code to run in the native machine language of the system on which it is executing. Meanwhile, the memory manager removes the possibilities of fragmented memory and increases memory locality-of-reference to further

increase performance.

Finally, the runtime can be hosted by high-performance, server-side applications. This infrastructure enables the use of managed code to write business logic.

.NET Framework Class Library

The .NET Framework class library is a collection of reusable types that integrate with the common language runtime. The class library is object oriented, providing types from which your own managed code can derive functionality. This not only makes the .NET Framework types easy to use, but also reduces the time associated with learning new features of the .NET Framework. In addition, third-party components can integrate seamlessly with classes in the .NET Framework.

For example, the .NET Framework collection classes implement a set of interfaces that you can use to develop your own collection classes. Your collection classes will blend seamlessly with the classes in the .NET Framework.

As you would expect from an object-oriented class library, the .NET Framework types enable you to accomplish a range of common programming tasks, including tasks such as string management, data collection, database connectivity, and file access. In addition to these common tasks, the class library includes types that support a variety of specialized development scenarios. For example, you can use the .NET Framework to develop the following types of applications and services:

- Console applications.
- Windows GUI applications (Windows Forms).
- Windows Presentation Foundation (WPF) applications.
- ASP.NET applications.
- Web services.
- Windows services.

Criticism

Some concerns and criticism relating to .NET include:

- Applications running in a managed environment tend to require more system resources than similar applications that access machine resources more directly.
- As JIT languages can be more easily reverse-engineered than native code to algorithms used by an application, there is concern over possible loss of trade secrets and the bypassing of license control mechanisms. Many obfuscation techniques already developed can help to prevent this; Microsoft's Visual Studio 2005 (and newer) includes such a tool.
- In a managed environment the regularly occurring garbage collection for reclaiming memory suspends execution of the application for an unpredictable lapse of time (typically no more than a few milliseconds, but in memory-constrained systems can be much longer). This makes such environments unsuitable for some applications such as those that must respond to events with predictable timing (such as in real-time computing).
- Since the framework is not pre-installed on older versions of Windows an application that requires it must verify that it is present, and if it is not, guide the user to install it. This requirement may deter some from using the application as the downloads is many megabytes in size.
- Newer versions of the framework (3.5 and up) are not pre-installed on any of the pre-Windows 7 versions of the Windows operating system. Some developers have expressed concerns about the large size (around 54 MB for end-users with .NET 3.0, 197 MB with .NET 3.5, and 250 MB with .NET 3.5 SP1) and reliability of .NET framework runtime installers for end-users. The first service pack for version 3.5 mitigates this concern by offering a lighter-weight client-only subset of the full .NET Framework.
- The .NET framework currently does not provide support for calling Streaming SIMD Extensions (SSE) via managed code. However, Mono has provided support for SIMD Extensions as of version 2.2 within the Mono.Simd namespace; Mono's lead developer Miguel de Icaza has

expressed hope that this SIMD support will be adopted by the CLR ECMA standard.[40] Streaming SIMD Extensions have been available in CPUs since the introduction of the Pentium III.

3.2 Selected Solution

After an initial period in which the software was developed in Borland Delphi, Visual Studio 2005 and the .NET Framework were the development platform selected for the final implementation of this project.

The reason for the change were:

1. the more user-friendly development environment provided by the Microsoft Visual Studio 2005;
2. versatility of the .NET Framework;
3. possibility to use the same code to generate a mobile version of the project for use in PDAs;
4. faster compiling than the Borland Delphi.

The Microsoft Visual Studio has improved considerably from previous versions, where it was surpassed by Borland Delphi. Later we discovered that this leap in quality coincided with the departure of Anders Hejlsberg from Borland to Microsoft, where he developed .NET and C [4] [1].

Chapter 4

Project Description

The Integrated Development & Management System that was created in this project aggregates all the development and management functionalities for the motronic system.

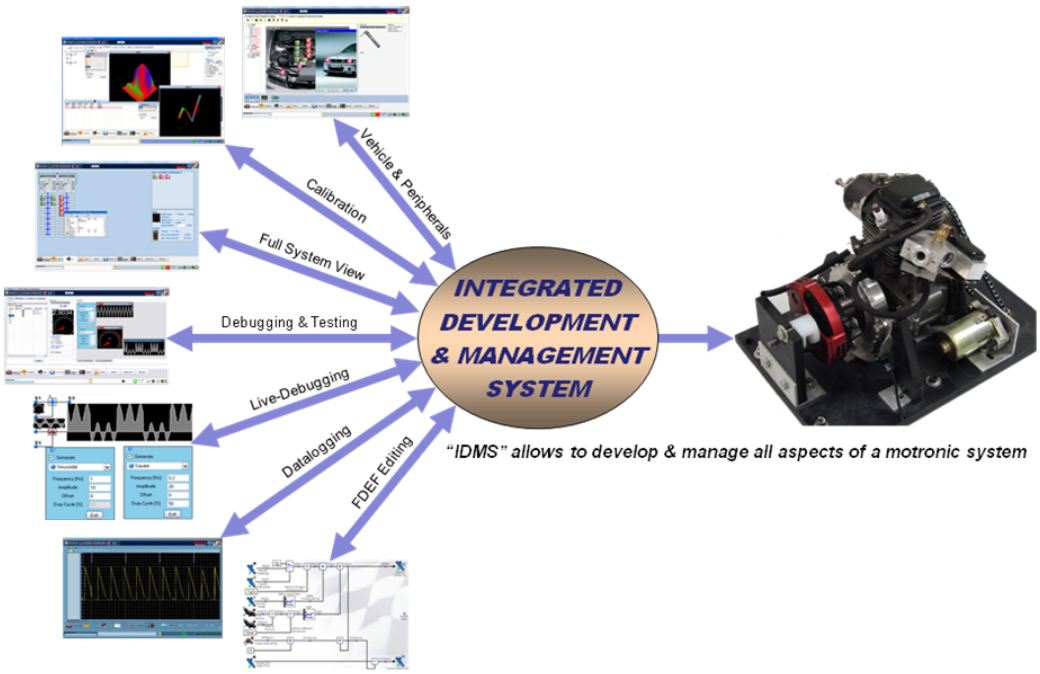


Figure 4.1: Integrated Development & Management System.

Since this software embraces so many different functionalities and options, it was subdivided into categories so that related functionalities are grouped together for easier development and maintenance.

The structure is shown in Figure 4.2.

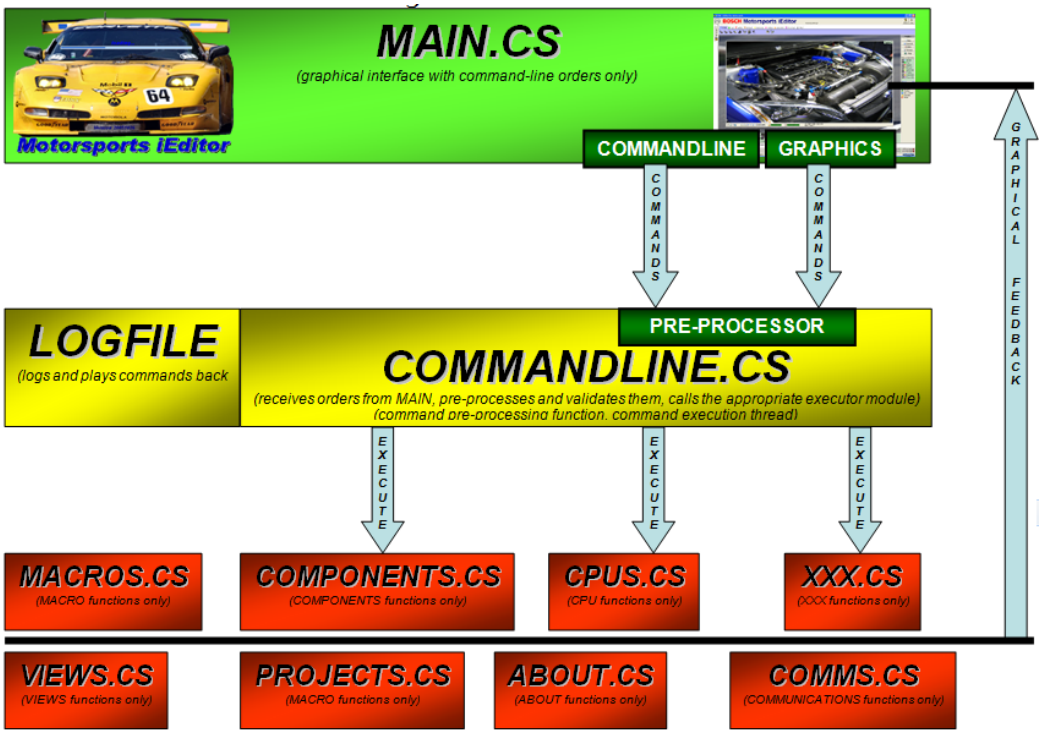


Figure 4.2: Integrated Development & Management Structure.

4.1 Overview

The layout of the IDMS is composed of: Main Menu, Work Area, Functionalities Tab Selector, Commandline and Status Bar. They integrate as show in Figure 4.3.

Figure 4.3: Integrated Development & Management Layout.

4.1.1 Main Menu

The main menu hosts the core options related to the project. It should be the starting point in the interaction with the program. The options' layout are described in the following sub-sections.

Project Menu

Contains the options to create, open, save or close a project. Additionally, it also includes the option to exit the IDMS. It's similar to the very common File Menu in the majority of Microsoft Windows Software.

Figure 4.4 shows the menu structure.

Actions Menu

This menu has the main actions in interacting with the hardware layer of this System.

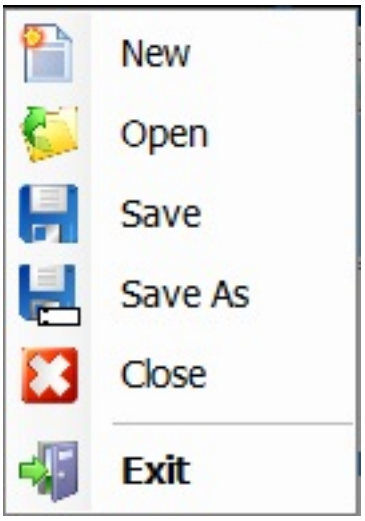


Figure 4.4: Project Menu.

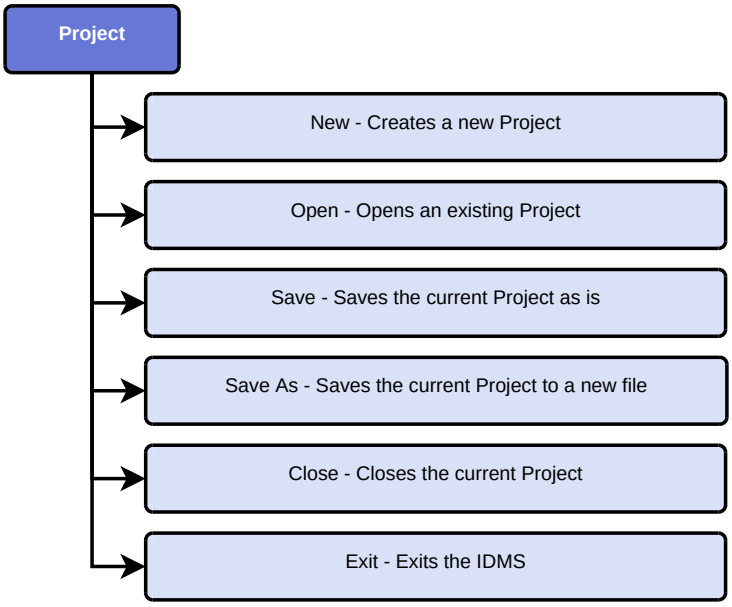


Figure 4.5: Project Menu diagram.

Figure 4.6 shows the menu structure.

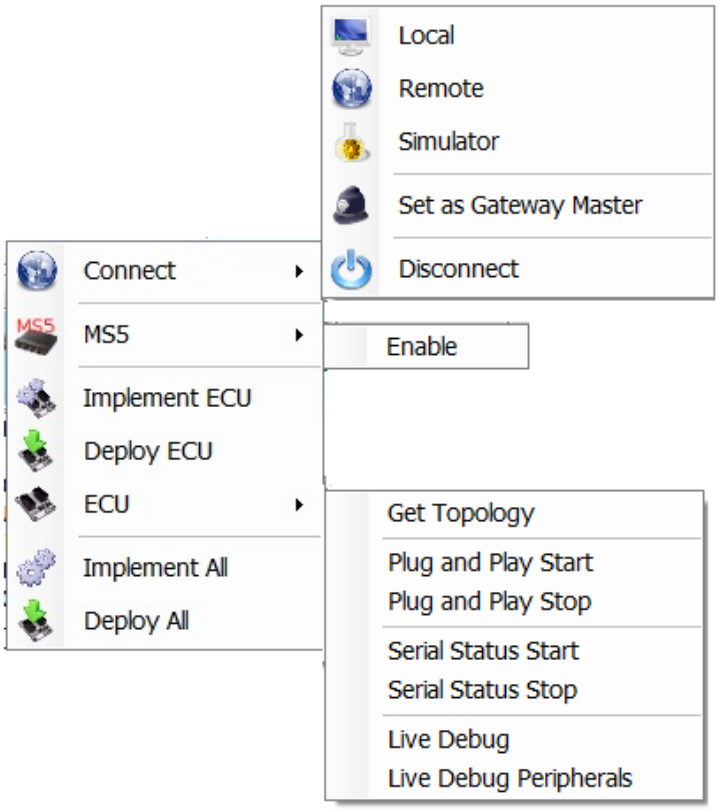


Figure 4.6: Actions Menu.

History Menu

The options present in this menu are related to the sequence of actions performed in using the IDMS. A log is stored and can be consulted here, as well as the options to undo or redo the performed action.

Figure 4.8 shows the menu structure.

Help Menu

The last menu in the main menu area is where the Help and IDMS's software info can be accessed. In this menu we can access the update and

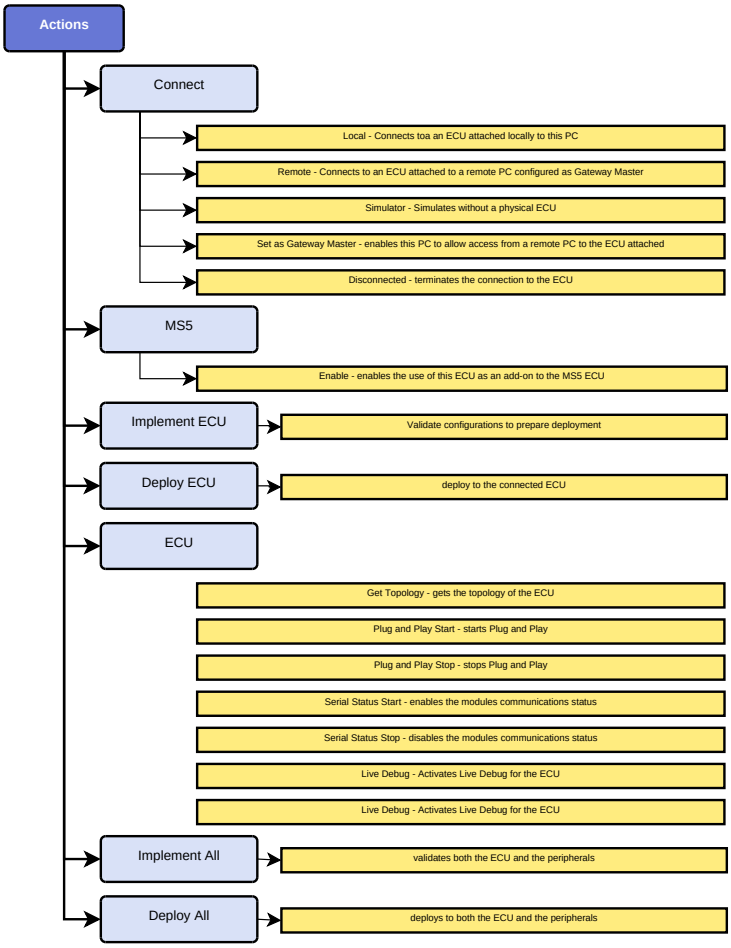


Figure 4.7: Actions Menu diagram.

problem reporting features.

Figure 4.10 shows the menu structure.

4.1.2 Command Line

The Command Line is similar to the one used in Microsoft Windows, only the command set available is restricted to the functionalities offered by the IDMS software.

All major actions performed in the IDMS can be done using either the

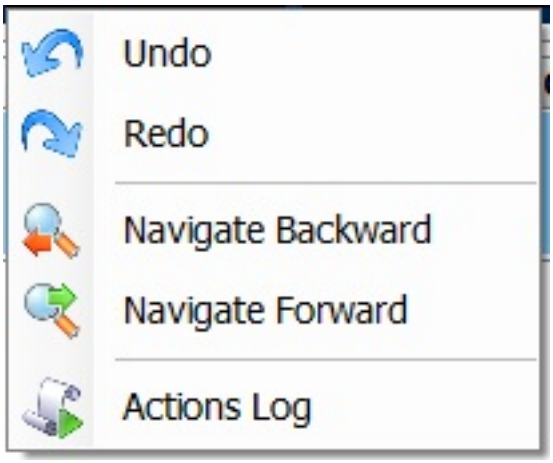


Figure 4.8: History Menu.

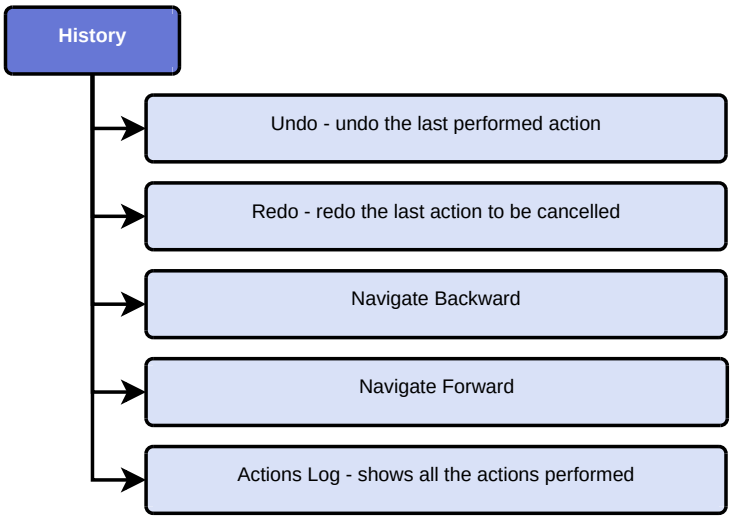


Figure 4.9: History Menu diagram.

command line or by graphical inputs. The graphical inputs are translated to command line instructions.

The command line instructions are analysed from left to right, where the first words define the category of the instructions and the remaining define the action. An example:

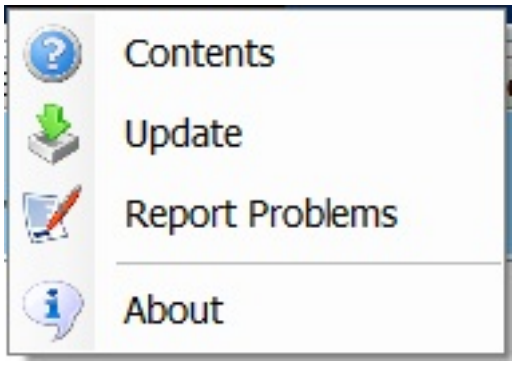


Figure 4.10: Help Menu.

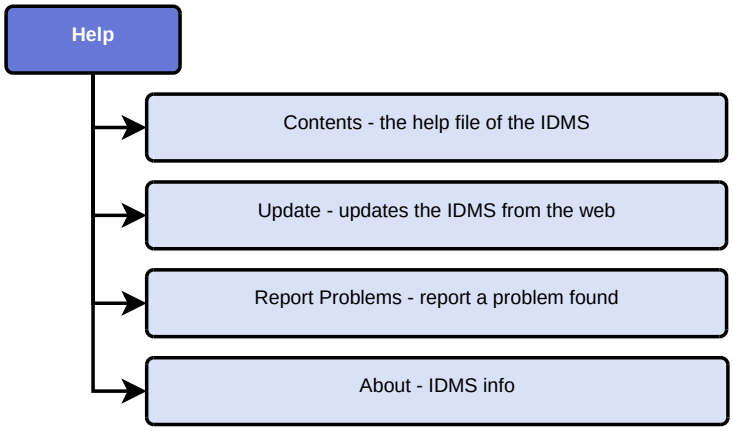


Figure 4.11: Help Menu diagram.

ADD MACRO MUL engineRev 10 outputVar

ADD MACRO defines the category (adding a new complex instruction to the current function). MUL is the complex instruction to add (multiplication), in which the variable engineRev is the first input, the constant 10 is the second input and the output is put in the variable outputVar. ADD MACRO can be replaced by AM for abbreviation.

Another example:

SHOW FUNCTION injectionQuantity

This command line instruction will show the function named injectionQuantity. SHOW FUNCTION can be replaced by SF for abbreviation.

A more experienced user can benefit from this functionalities, but there are more advantages: it's easier to keep a history of the actions performed and even import/export a sequence of instructions to perform in other instance of the IDMS (similar to macros).

4.1.3 Work Area

The content of this Area changes accordingly to the option chosen from the Functionalities Tab Selector.

These functionalities are:

- Vehicle Views;
- Functions Editor;
- ECU Management;
- Project Lists;
- Datalog Viewer;
- Display;
- Extras;

It's in the area that the real development and management work will occur.

4.2 Vehicle Views

The Idea: to have a visualization of the placement of the peripherals in the vehicle, while having access to complete information and interaction with them.

Advantages: when an error occurs, it's quick to know exactly which of the peripherals needs assistance and see the location on the vehicle. Any action can be directly performed by simply selecting the appropriate peripheral and option.

Functionalities

Multiple vehicle views: it's possible to have multiple different views of the vehicle. The only necessary item is an image file with the required vehicle

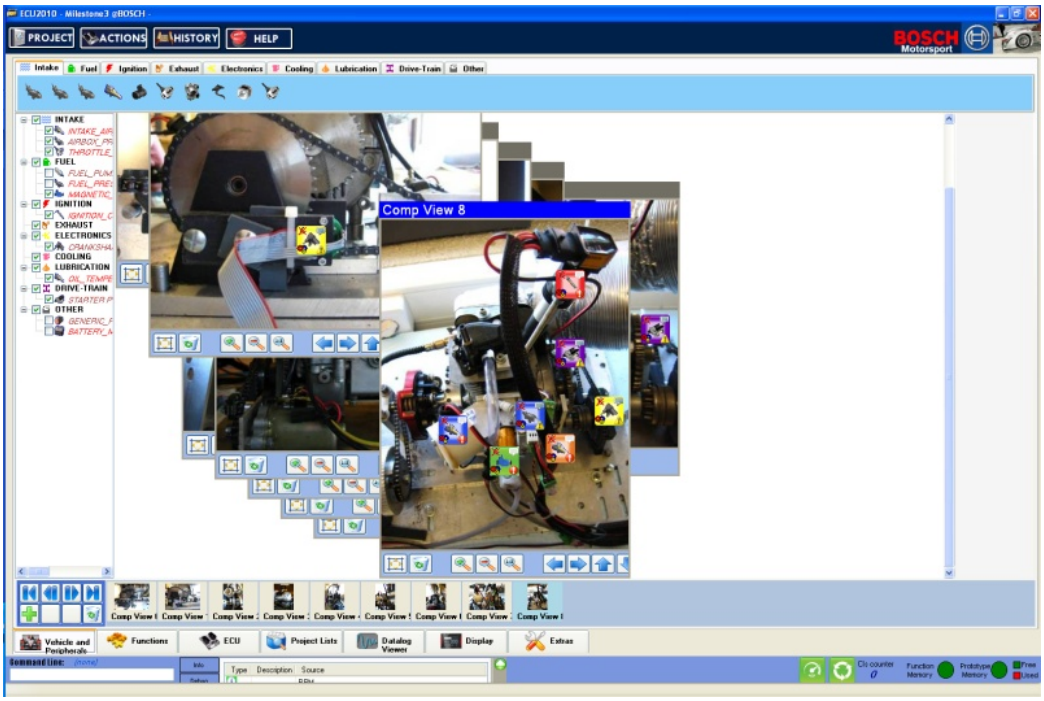


Figure 4.12: Vehicle and Peripherals Views.

perspective. After the new view is created, it's possible to drag and drop peripherals icons to the intended place in the view. This icons show all the necessary info related to the peripheral (status, type, etc.).

Jumping to functions: by right clicking in a peripheral, it's possible to jump to any of the functions in which any of the peripheral's variables are used.

Jump to views: again, using the context menu available through the right click, it's possible to jump to any view in which the selected peripheral is represented.

Assign Plug: directly assign the peripheral to a specific plug in a given module.

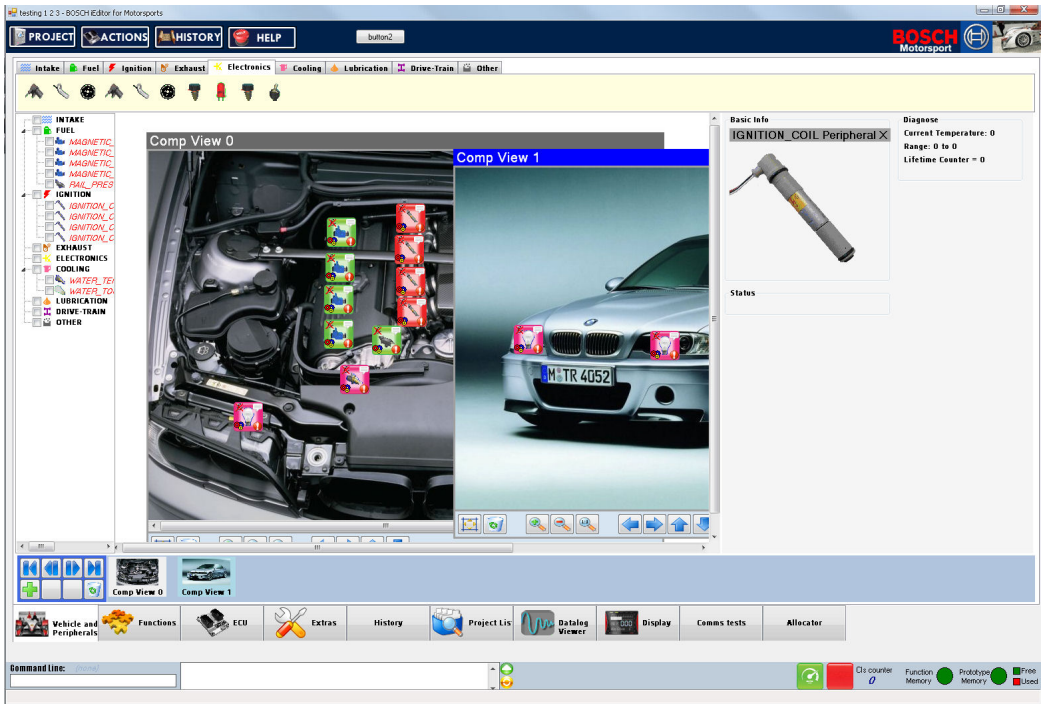


Figure 4.13: Multiple Views and Peripheral Info.

4.3 Functions Editor

The Idea: create a visual editor to create and modify all the project's functions. The intelligence of the of the motronic system is based on the functions it is running, so this is a vital part of the development process. The main goal to achieve here is to create an environment in which the user can edit the functions without the need to know a programming language or to use a manual. All that is required is to know what data to read, how to process it and what to actuate. Obviously, this led to an approach in that each function is automatically represented in a visual sequence. This sequence can be directly modified, adding, removing or altering the complex instructions that are the building blocks of the functions.

The concepts to grasp are Node and Complex Instructions. A Node is a variable, with a unique ID number, is stored in the BCDP (Binary Coded Decimal Power) format in the GIMy (Gateway Intelligent Memory). A Com-

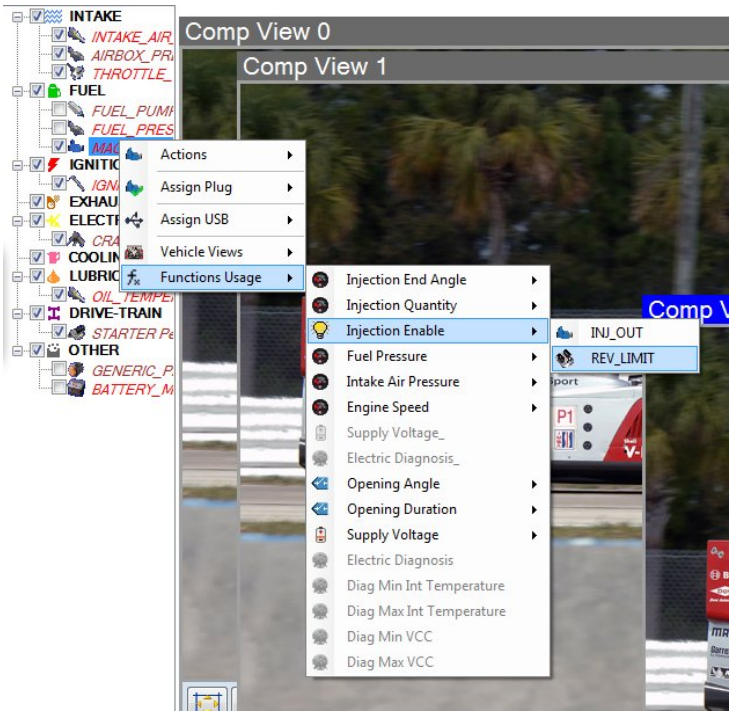


Figure 4.14: Jump to Function.

plex Instruction is a atomic operation (add, multiply, logical or, logical and, low pass filter,...)The functions are a logic sequence of Complex Instructions.

The programmer doesn't need to worry about the visual alignment of the Complex Instructions. The only concern should be at the logical level, as to were to place the CI to have the expected result. Every action the programmer does in the Editor will be translated into changes in the Function code. After these changes occur the function will be redrawn in the work area of the Functions Editor. It's the code sequence of the CIs that is represented visually and not the other way around.

Functionalities

Functions Explorer: Lists all the functions in the project, giving the possibility to add new functions, modify existing ones and change its status. Quick and easy way to jump between functions.

Nodes: the nodes are the data storage units in the functions. They serve as inputs and/or outputs to the Complex Instructions that constitute the

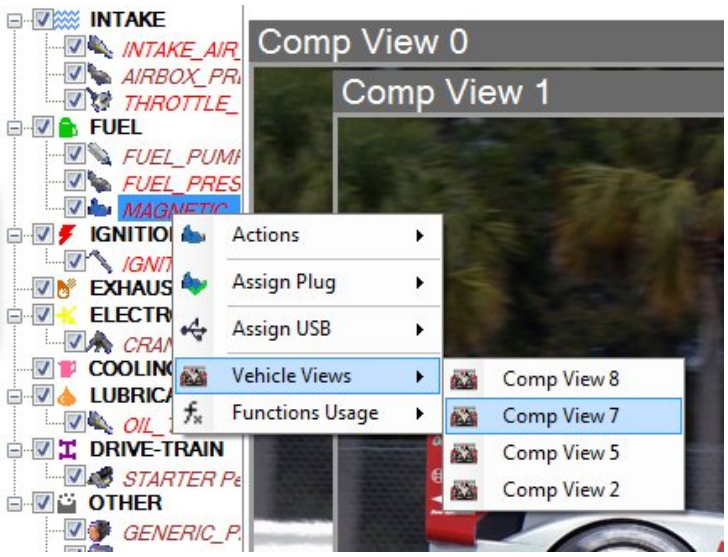


Figure 4.15: Jump to View.

functions. Each node has a visual representation in the functions in which they are used.

Complex Instructions: each function is formed by a logical sequence of Complex Instructions (CI). These CIs are atomic operations, with multiple inputs and at least one output. These operations can be as basic as adding, multiplying or more complex, like a filter. Any CI can be resumed in a line of code. For example, adding 5 to node in1 and putting the result in out1 is as simple as ADD in1 5 out1

Visual Editing: all the operations in the functions can be done in a completely visual mode.

Projects: The complete solution (functions, peripherals, configurations, etc) can be saved as a Project. This project can be loaded in any IDMS to make changes or interact with the saved configuration. The whole project is saved as an XML file. Despite creating a bigger file when compared to a binary file, it has many advantages such as being human readable and editable.

Comparing Functions: it provides a visual representation of the differences between functions (Complex Instruction that were added, removed or modified). Despite the visual representation, all the changes are calculated

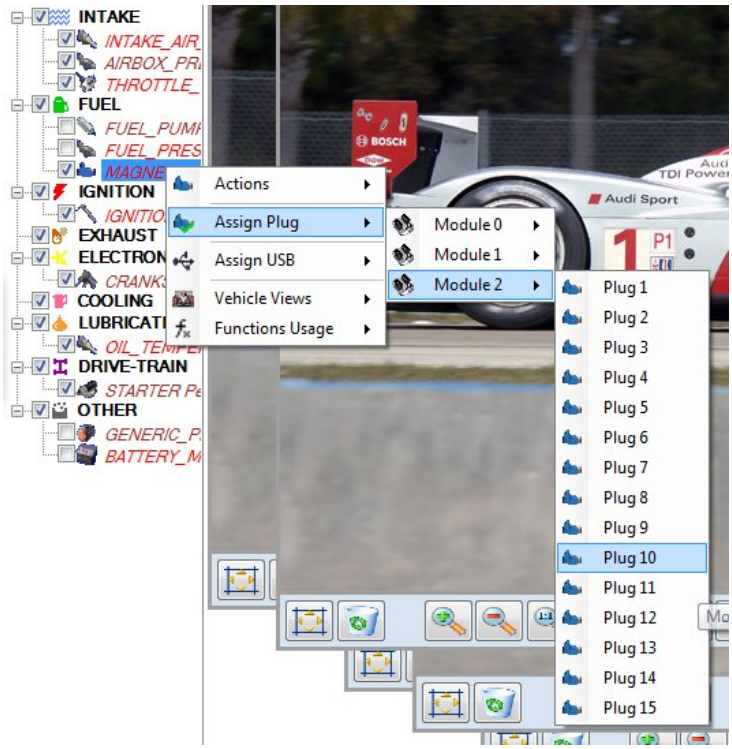


Figure 4.16: Assign Plug.

from the simple code representation of the Complex Instructions sequence of the functions.

A simple example:

Function A
1. ADD var1 var2 out1
2. MUL out1 8 out2
3. SUB out2 4.45 out3

Function B
1. ADD var1 var2 out1
2. SUB out2 4.8 out3
3. DIV out3 2 out4

We can easily see that from Function A to Function B the differences are: code line 2 was removed from Function A; the code line 3 was changed (constant value changed from 4.45 to 4.8); a new code line was added to the end of the function. All this changes (detected after the analysis) can then be properly flagged so that upon redrawing this can be perceived.

This component (Functions Editor) was the single most time consuming feature to develop.

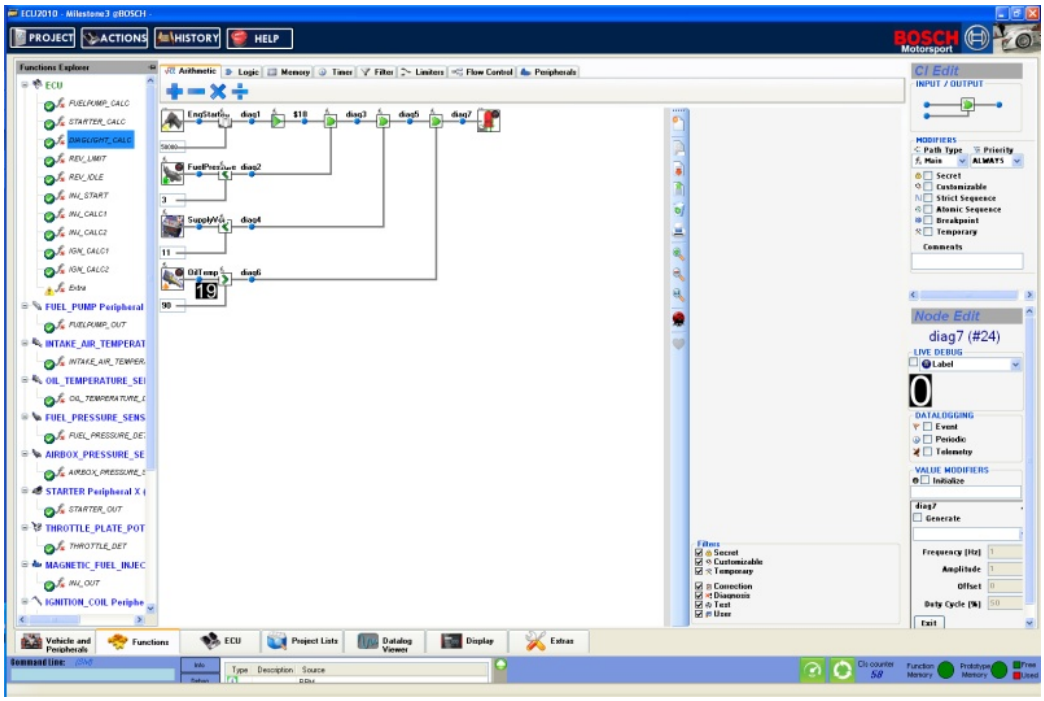


Figure 4.17: Functions Editor.

4.4 ECU Management

The Idea: to have a visual representation of all the modules (Cellular ECU) used in the ECU, for a quick visualization of the resources used and the limitations of the hardware as is. The working environment displays the Cellular ECUs with their reference ID and status, the connections between them and their status, the peripherals that are connected to each Cellular ECU, the peripherals that aren't assigned as well as the maximum specs that the current configuration can support (RPMs, power requirements, etc.).

Functionalities

Plug Play: As the peripherals are connected/disconnected, these changes are immediately visible in the visual representation. The peripheral is detected, as well as it's plug, type and additional data.

Module occupancy: the modules being used and the occupancy in each one

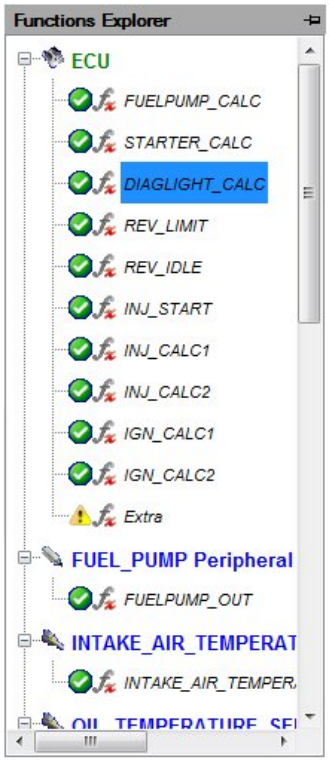


Figure 4.18: Functions explorer.

4.5 Project Lists

The Idea: to have lists with direct access to all the elements in the project (nodes, functions, modules and peripherals). This was very useful in the development process.

Functionalities

Nodes List: see all the nodes in the project either all at once in a flat list or as a hierarchy tree in which we can see the variables in the ECU or in each peripheral. All the functionalities for nodes available in the Function Editor are also available here with the addition of a monitoring area where we can add multiple visualizations of the nodes' values.

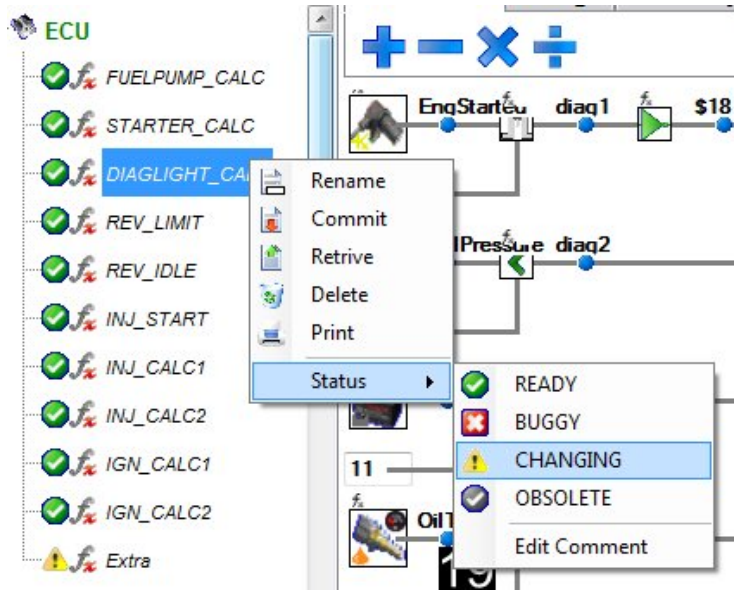


Figure 4.19: Functions Explorer's Context Menu.

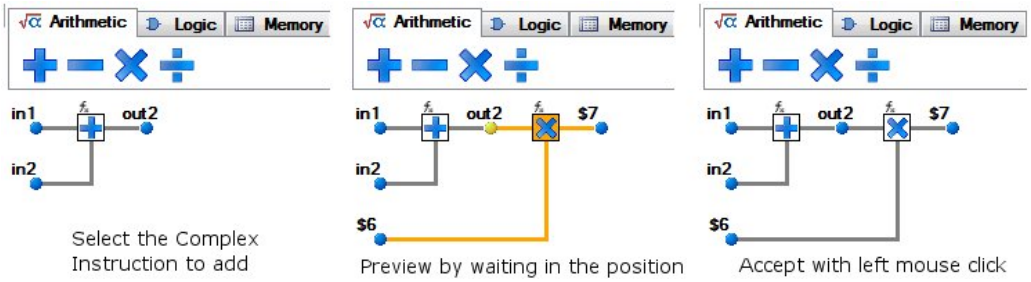


Figure 4.20: Adding a new Complex Instruction (Live Dragging).

4.6 Datalog Viewer

The Idea: to analyse the data recorded while the system was operating. This data can be visualized in an interactive graphical view with pan and zooming functionalities.

In motorsports, just like in every sport, in order to progress one must evaluate their own results.

Typically this functionality is provided by a standalone software, but as

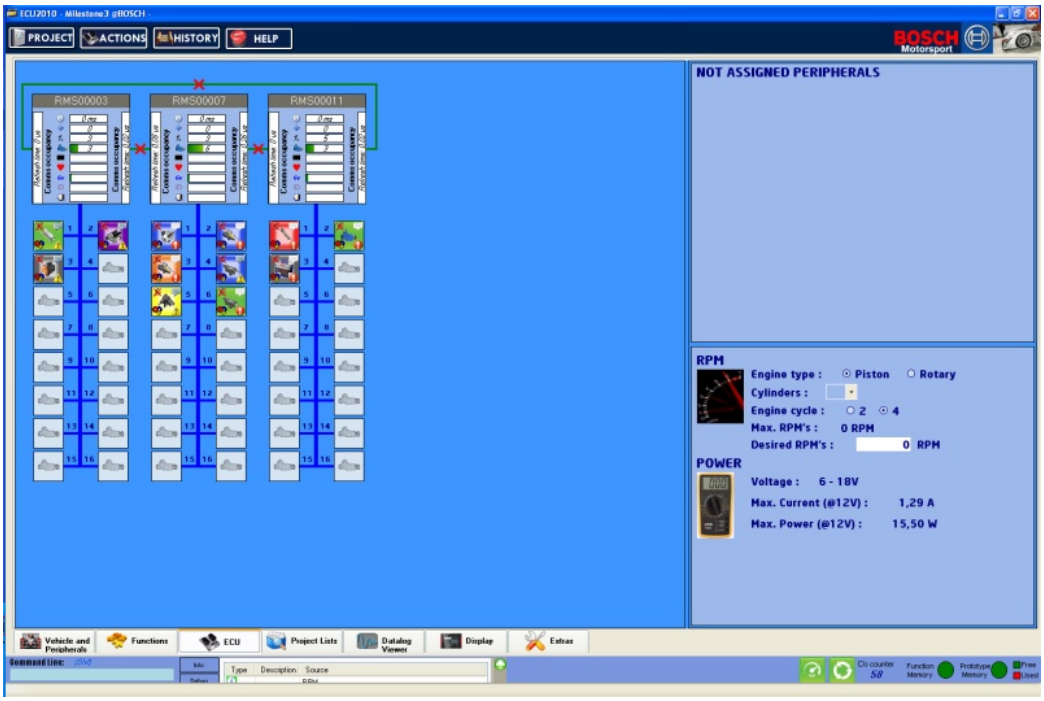


Figure 4.21: ECU Management.

a key feature in motorsports, it had to be integrated into our solution.

4.7 Display

In motorsports there is a serious need for the use of digital displays for an easier and flexible access to specific data available to the ECU so that it can be available to the pilot during the race/trainings.

This project approach to the display takes it some steps forward, giving them the possibility to have all the functionalities of the IDMS or a more constrained range of options.

The "Display" section in the IDMS is where i's possible to view and design the dashboard display components for the drivers in a "What You See Is What You Get" fashion. We can drag-and-drop different kinds of visualization controls that can be associated to specific nodes.

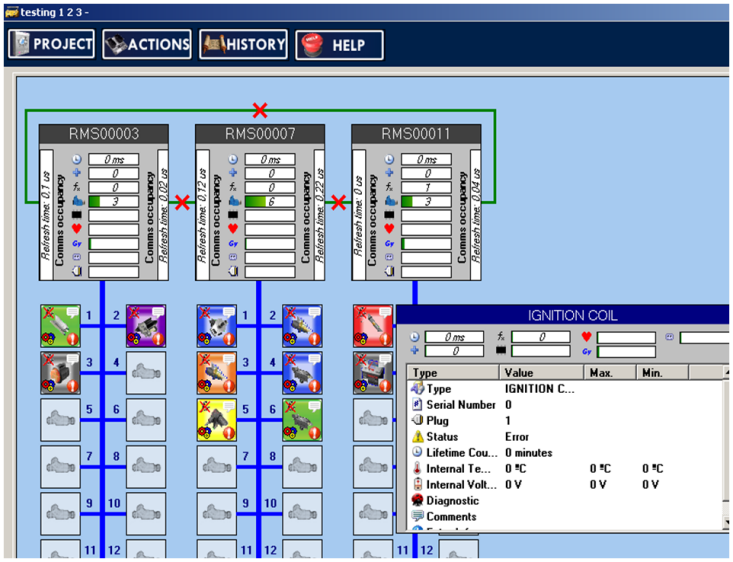


Figure 4.22: Module occupancy, plug play and peripheral info.

4.8 Extras

This section of the IDMS is where the remaining miscellanea of tools are arranged.

At this point it's possible to:

- Visit the Bosch Motorsport's homepage (integrated basic web browser using a control available in Microsoft Visual Studio);
- Consult the Catalogue to see the available components, it's characteristics and price;
- Actions Log the history of all the actions performed in the IDMS.

At this point it's still not possible to:

- See Tele Operation options and logs in this tab, only accessible in development form;
- Version Control;

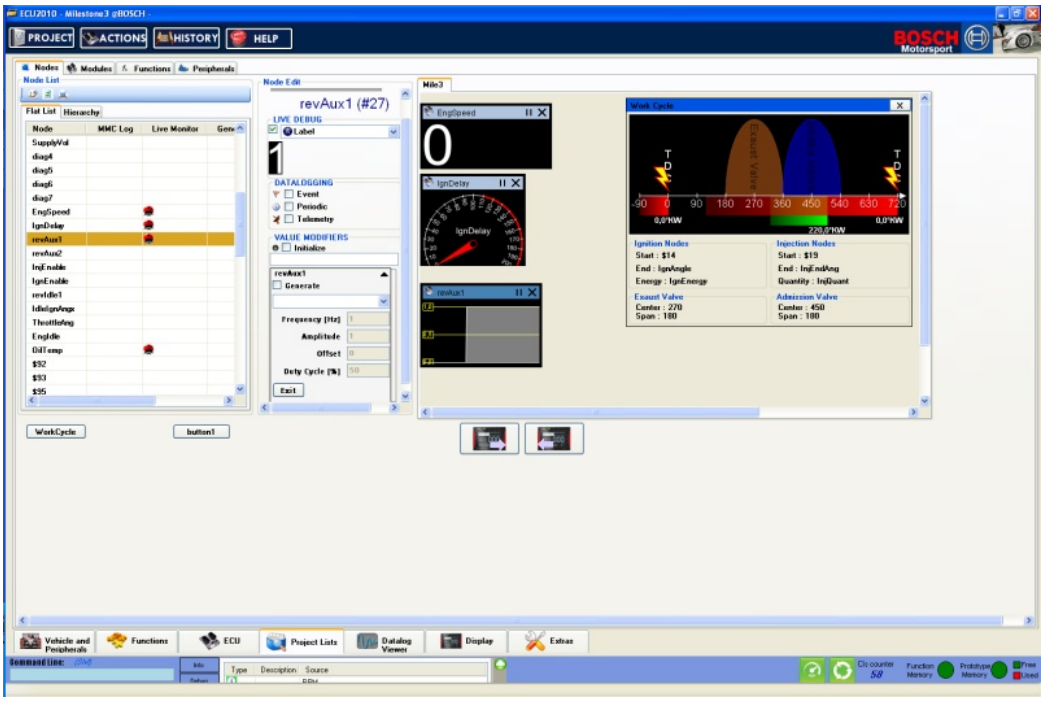


Figure 4.23: Project Lists.

- Calculate the system cost it implies access to an online database with all the references and prices. Not done yet;
- Report changes the ability for the users to directly request modifications and added functionalities to the IDMS.

4.9 Updates

To keep simplicity as possible, the updates of new versions of the IDMS only require the user to acquire the new version executable file and start using it instead of the old version.

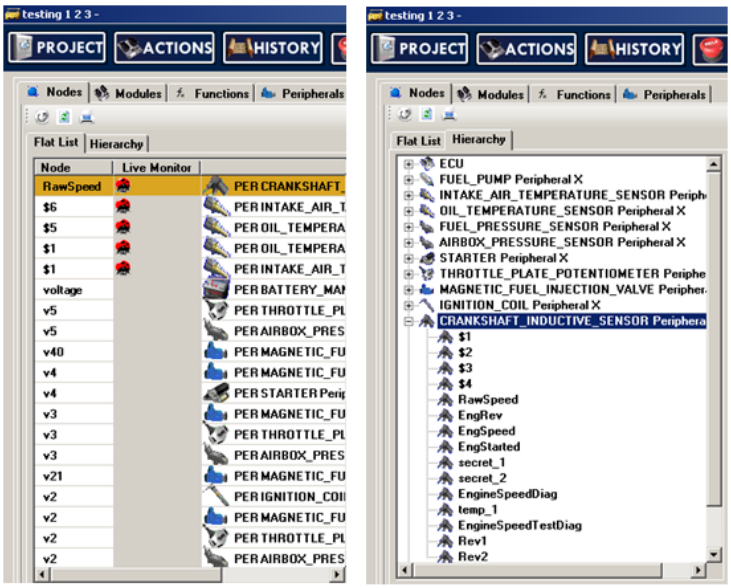


Figure 4.24: Nodes Lists.

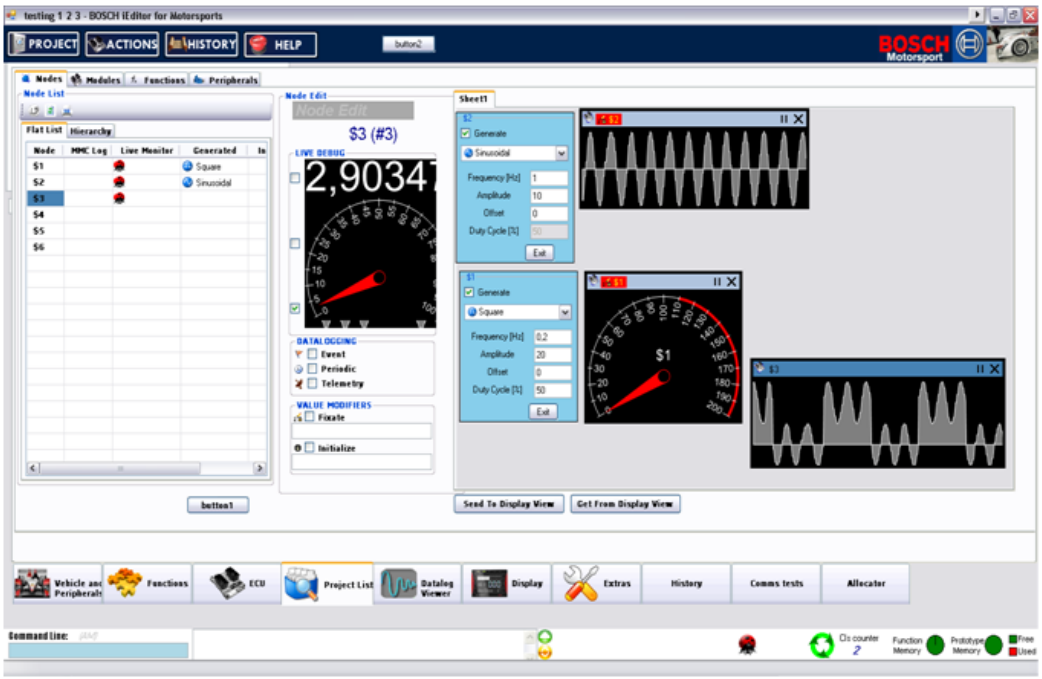


Figure 4.25: Nodes' values monitoring.

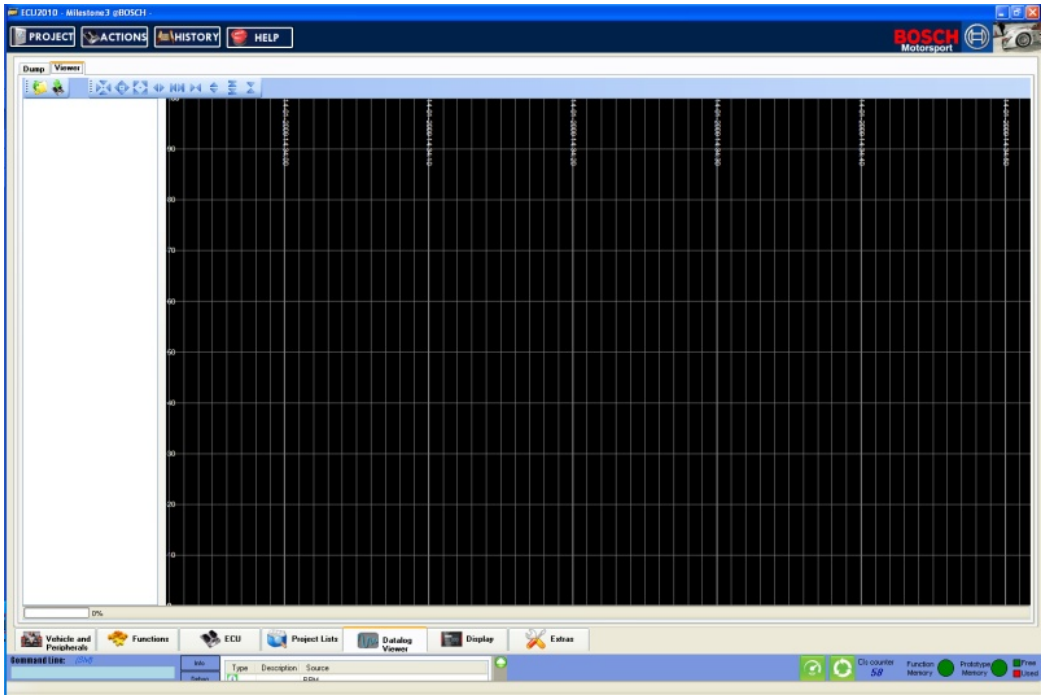


Figure 4.26: Datalog Viewer.

Figure 4.27: Data representation.

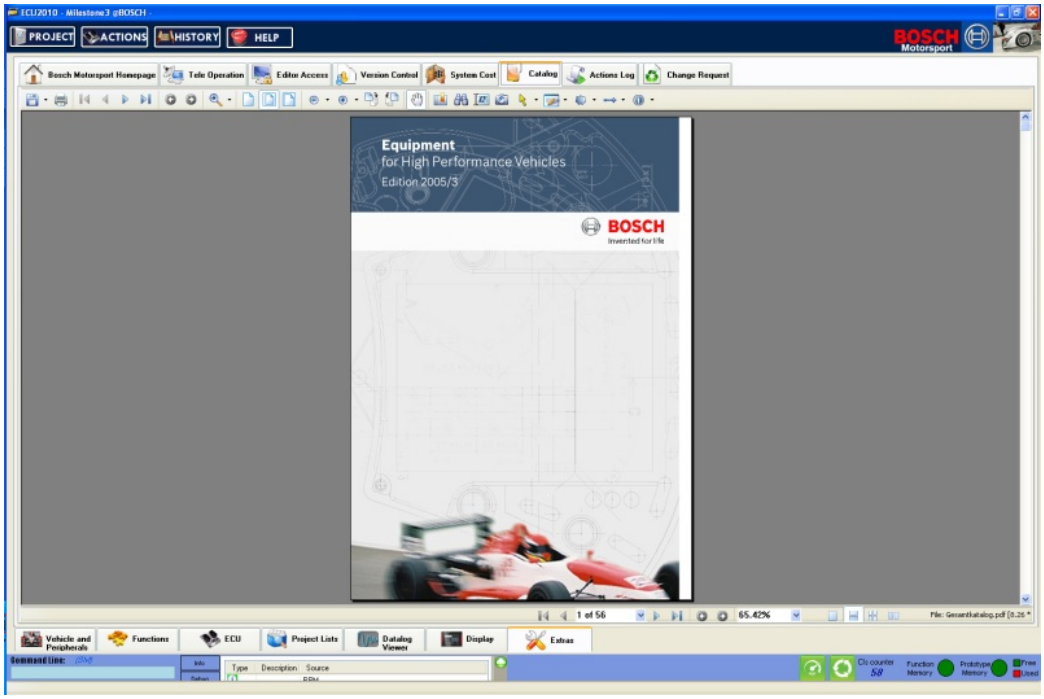


Figure 4.28: Extras.

Chapter 5

Results

5.1 Milestone 1

The first time the IDMS was demonstrated was in 29 of March of 2007 at the Milestone 1 of the project ECU2010.

The demonstrations where performed at University of Aveiro to a small group of representatives from Bosch Motorsports and Robert Bosch GmbH.

At this point, the IDMS only had functionalities related to the Functions Editor and didn't interact directly with all the other project components. It was possible to show:

- Adding Complex Instruction both visually and by commandline;
- Function comparison;
- Live dragging;

5.2 Milestone 2

The Milestone 2 in 15 of September of 2008 was a preparation for Milestone 3, in which were presented:

- Integration of all developed Hardware and Software components;
- Tele-operation;
- Live-prototyping;

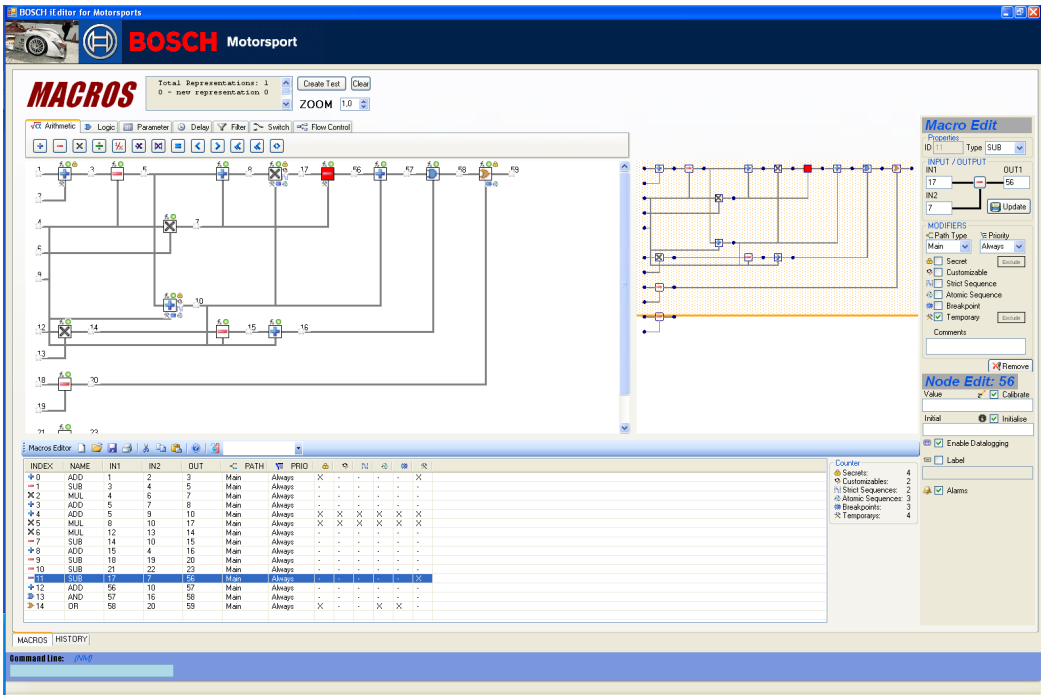


Figure 5.1: Milestone 1: Functions Editor.

- Live-debugging;
- Plug and Play on Smart Peripherals;
- Wireless Datalogger;
- Vehicle and Peripherals views features.

5.3 Milestone 3

In 28 of November of 2008, in a presentation in Stuttgart, Germany, at the Bosch Motorsport building, the Milestone 3 of the project was presented.

The presentation schedule consisted in an initial global presentation of the Project, followed by Questions/Answers and a more detailed technical explanation.

This presentation started with an overview of the project motivation, principles and goals.

An internet connection was established between Bosch Motorsport in Stuttgart, Germany and University of Aveiro, Portugal. A monocilinder 4 stokes engine was in Aveiro, controlled by a prototype of the new ECU, connected to a PC running the IDMS software. In Germany, we runned the same IDMS software in a Laptop connected to the internet and a link was established to the PC in Portugal. From this point on, all the actions were executed thru the IDMS in Germany, managing the engine in Portugal. A test project was then loaded and the engine was started remotely with IDMS, using the Functions Editor with a function in which the fuel pump and the ignition were activated.

To show the live-prototyping functionality and with the engine running, we proceeded to introduce some changes to a function. For the sake of visibility of the changes operated, we proceeded to modify the injection calculus function and increased the fuel quantity injected into the engine cylinder. This resulted in an immediate increase in the engine revs that could both be heard in the engine and seen in the IDMS in the node that contained the value received from the rpm sensor in the engine.

Once again, we changed the injection calculus function to reduce the fuel quantity injected and the engine responded with a decrease of the rev count.

We also demonstrated the plug-and-play capabilities by removing a peripheral from the plug it was connected and connecting it to another plug in other Cellular ECU. All this process had repercussions in the IDMS, either at the ECU Management tab, where the placement and characteristics of the peripheral could be perceived, or at the Functions Editor, where the peripheral variables can be used to construct the functions.

The Milestone 3 was a success, with all the intended highlights proven functional.

Chapter 6

Conclusions

The project (as a whole) was a success. The mains features were demonstrated and are functional, even if it stills needs to be polished to be more reliable.

Even if this software doesn't appear to be really revolutionary in respect to the available actions, it is a giant step forward in the motorsport industry to have all the needed software tools integrated into this IDMS, with all features designed from ground up specifically to this purpose.

From the moment a user starts to conceive the ECU, through the intelligent peripheral development, data analysis, functions creation or even checking the working status of the hardware, only this IDMS is needed. This integration is what gives the edge to user, because all actions are interconnected and the results are immediate.

The functions editor was the single most time consuming feature of this project's software component. As new Complex Instructions where added and new functionalities were conceived, it was necessary to take some steps backs and change the code to support them. The largest drawback of the approach taken to graphically represent the functions is the use of an object derived from the System.Windows.Forms.Button object to represent each of the Complex Instructions and the use of an object derived from the System.Windows.Forms.Control object to represent the connections between the nodes and the Complex Instructions. This means that each time we require that a function is shown in the editor, all the graphical objects need to be created and this is time and resource consuming. Even so, this was the solution chosen because of the easier access to point-and-click functionalities, mouse-over events and right-click menus.

6.1 Future Work

The next step would be to continue the work and develop into a ready for commercial status. This will be a time consuming stage with an emphasis in debugging and optimization, since all the main functionalities have already been proved.

The graphical Function Editor would need some serious rethinking, as the approach taken is not very flexible nor performant and is very limited in the visual effects capabilities.

It's essential for the software to continue to operate in error events so that any harmful action can be undone to a working state, no damage occurs to the hardware and the project file can always be recovered.

Chapter 7

Glossary

A

API Application programming interface.

C

CI Complex Instruction. It's the atomic constituent element of the Functions. A CI can be seen as an operation.

CLR Common Language Runtime

Compiler - A Compiler is a software that transforms source code into object files.

CTS Common Type System

E

ECU Electronic Control Unit. It's a generic term for any embedded system that controls one or more of the electrical systems or subsystems in a motor vehicle.

ESDL Embedded Software Description Language. Programming language for Embedded Systems, supported by ETAS tool ASCET.

F

FDEF Function Definition. See Function.

FPGA Field-Programmable Gate Array. Integrated circuit that can be programmed after manufacturing [6].

Function Sequence of Complex Instructions with a specific purpose.

I

IDMS Integrated Development and Management System. It's this project result: Software that integrates all the tools needed in a revolutionary approach to the Motorsport ECUs.

IIS Internet Information Services

J

JIT Just-In-Time.

L

Linker - a Linker combines the object files generated by the compiler into a single executable program.

N

Node it's a variable addressed in memory which can be read and/or written by the Complex Instructions or Peripherals.

P

PDA Personal Digital Assistant.

Bibliography

[1] Anders hejlsberg in wikipedia. http://en.wikipedia.org/wiki/Anders_Hejlsberg.

[2] Ascet definition in wikipedia. http://de.wikipedia.org/wiki/ASCET.

[3] Ascet official site. http://www.etas.com/en/products/ascet_software_products.php.

[4] Borland history. http://en.wikipedia.org/wiki/BorlandBorland_reborn_in_name_and_fame.

[5] Delphi definition in wikipedia. http://en.wikipedia.org/wiki/Embarcadero_Delphi.

[6] Fpga definition in wikipedia. http://en.wikipedia.org/wiki/Field-programmable_gate_array.

[7] Labview official site. http://www.ni.com/labview/.

[8] Mathworks official site. http://www.mathworks.com/products/matlab/.

[9] .net in microsoft.com. http://www.microsoft.com.

[10] W3c recommendation for xml (extensible markup language) site. http://www.w3.org/TR/REC-xml/.

[11] Xml definition in wikipedia. http://en.wikipedia.org/wiki/XML.